



# **Reed-Solomon II MegaCore Function**

---

## **User Guide**



101 Innovation Drive  
San Jose, CA 95134  
[www.altera.com](http://www.altera.com)

UG-01090-2.0

Document last updated for Altera Complete Design Suite version:  
Document publication date:

11.0  
May 2011



[Subscribe](#)

© 2011 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX are Reg. U.S. Pat. & Tm. Off. and/or trademarks of Altera Corporation in the U.S. and other countries. All other trademarks and service marks are the property of their respective holders as described at [www.altera.com/common/legal.html](http://www.altera.com/common/legal.html). Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



## Chapter 1. About This MegaCore

Features .....	1-1
Device Family Support .....	1-1
MegaCore Verification .....	1-2
Performance and Resource Utilization .....	1-3
Release Information .....	1-4

## Chapter 2. Getting Started

Installation and Licensing .....	2-1
Evaluating an IP Core .....	2-2
Open Core Plus Time-Out Behavior .....	2-2
MegaWizard Plug-In Manager Design Flow .....	2-3
Specifying Parameters .....	2-3
Simulating the Design .....	2-4
Compiling the Design and Programming a Device .....	2-4
Parameter Settings .....	2-5

## Chapter 3. Functional Description

Architecture .....	3-1
Interfaces .....	3-1
Avalon-ST Input and Output Interfaces .....	3-2
Clock and Reset Interfaces .....	3-2
Status Interface .....	3-2
RS II Encoder .....	3-2
RS II Decoder .....	3-3
Multi-Channel Codeword .....	3-5
Signals .....	3-7

## Appendix A. Reed-Solomon Codes

RS Encoding .....	A-1
Field polynomial .....	A-1
Generator polynomial .....	A-2
Shortened Codewords .....	A-2
RS Decoding .....	A-3
Syndrome Polynomial .....	A-3
Error Polynomials .....	A-3
Error Location and Error Value .....	A-4

## Additional Information

Document Revision History .....	Info-1
How to Contact Altera .....	Info-1
Typographic Conventions .....	Info-1

The Altera Reed-Solomon (RS) II MegaCore® function comprises a fully parameterizable high-speed parallel encoder and decoder for forward error correction applications. RS codes are widely used for error detection and correction in a wide range of DSP applications for storage, retrieval, and transmission of data. The MegaCore function supports multiple channels that reduces resource usage and increases throughput.

## Features

The RS II MegaCore function supports the following features:

- High-performance encoder or decoder for error detection and correction
- 1.28 Gbps per channel for Altera 40 nm device families (Arria® II)
- Fully parameterized RS II MegaCore functions, including:
  - Number of symbols per codeword
  - Number of check symbols per codeword
  - Field polynomial
  - Multi-channel codeword
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators

## Device Family Support

Table 1–1 defines the device support levels for Altera IP cores.

**Table 1–1. Altera IP Core Device Support Levels**

FPGA Device Families	HardCopy® Device Families
<b>Preliminary support</b> —The IP core is verified with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.	<b>HardCopy Companion</b> —The IP core is verified with preliminary timing models for the HardCopy companion device. The IP core meets all functional requirements, but might still be undergoing timing analysis for the HardCopy device family. It can be used in production designs with caution.
<b>Final support</b> —The IP core is verified with final timing models for this device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.	<b>HardCopy Compilation</b> —The IP core is verified with final timing models for the HardCopy device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.

Table 1–2 shows the level of support the RS II MegaCore function offers to each of the Altera device families.

**Table 1–2. Device Family Support**

Device Family	Support
Arria GX	Final
Arria II GX	Final
Arria II GZ	Final
Cyclone® II	Final
Cyclone III	Final
Cyclone III LS	Final
Cyclone IV GX	Final
HardCopy II	HardCopy Compilation
HardCopy III	HardCopy Compilation
HardCopy IV E	HardCopy Compilation
HardCopy IV GX	HardCopy Compilation
Stratix®	Final
Stratix GX	Final
Stratix II	Final
Stratix II GX	Final
Stratix III	Final
Stratix IV GT	Final
Stratix IV GX/E	Final
Stratix V	Preliminary
Other device families	No support

## MegaCore Verification

Before releasing a version of the RS II MegaCore function, Altera runs comprehensive regression tests to verify its quality and correctness. Altera generates custom variations of the RS II MegaCore function to exercise the various parameter options and thoroughly simulates the resulting simulation models with the results verified against master simulation models.

## Performance and Resource Utilization


 Arria II GX, Cyclone III, and Stratix III devices use combinational adaptive look-up tables (ALUTs) and logic registers.

Table 1–3 shows the typical performance for Arria II GX (EP2AGX45DF2513) device using the Quartus® II software.

**Table 1–3. Performance—Arria II GX Devices**

Parameters					ALUTs	Logic Registers	Memory (M9K)	f <sub>MAX</sub> (MHz)
Options	Variant	Field Polynomial	Symbols per codeword ( <i>M</i> )	Check symbols per codeword ( <i>R</i> )				
Encoder	Channel 1	285	255	16	167	156	0	463
	Channel 2	285	255	16	168	287	0	481
	Channel 8	285	255	16	187	47	4	364
Decoder	Channel 1	285	255	16	1,537	939	1	278
	Channel 2	285	255	16	1,579	1,748	2	306
	Channel 8	285	255	16	1,663	877	24	271

Table 1–4 shows the typical performance for Cyclone III (EP3C5F256C6) device using the Quartus II software.

**Table 1–4. Performance—Cyclone III Devices**

Parameters					ALUTs	Logic Registers	Memory (M9K)	f <sub>MAX</sub> (MHz)
Options	Variant	Field Polynomial	Symbols per codeword ( <i>M</i> )	Check symbols per codeword ( <i>R</i> )				
Encoder	Channel 1	285	255	16	202	156	0	353
	Channel 2	285	255	16	314	287	0	308
	Channel 8	285	255	16	218	44	4	245
Decoder	Channel 1	285	255	16	2,292	934	1	183
	Channel 2	285	255	16	2,928	1,740	2	149
	Channel 8	285	255	16	2,410	861	24	159

Table 1–5 shows the typical performance for Stratix III (EP3SL50F484C2) device using the Quartus II software.

**Table 1–5. Performance—Stratix III Devices (Part 1 of 2)**

Parameters					ALUTs	Logic Registers	Memory		f <sub>MAX</sub> (MHz)
Options	Variant	Field Polynomial	Symbols per codeword ( <i>M</i> )	Check symbols per codeword ( <i>R</i> )			M9K	M144K	
Encoder	Channel 1	285	255	16	166	156	0	0	521
	Channel 2	285	255	16	167	287	0	0	516
	Channel 8	285	255	16	188	47	4	0	480

**Table 1–5. Performance—Stratix III Devices (Part 2 of 2)**

Parameters					ALUTs	Logic Registers	Memory		f <sub>MAX</sub> (MHz)
Options	Variant	Field Polynomial	Symbols per codeword (M)	Check symbols per codeword (R)			M9K	M144K	
Decoder	Channel 1	285	255	16	1,519	937	1	0	365
	Channel 2	285	255	16	1,585	1,754	2	0	337
	Channel 8	285	255	16	1,668	877	24	0	332

## Release Information

Table 1–6 provides information about this release of the RS II MegaCore function.

**Table 1–6. RS II Compiler Release Information**

Item	Description
Version	11.0
Release Date	May 2011
Ordering Codes	IP-RSCODECII (Primary License) IPR-RSCODECII (Renewal License)
Product IDs	00E5 (Encoder/Decoder)
Vendor ID	6AF7



For more information about this release, refer to the [MegaCore IP Library Release Notes and Errata](#).

Altera verifies that the current version of the Quartus II software compiles the previous version of each MegaCore function. The [MegaCore IP Library Release Notes and Errata](#) report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release.

This chapter provides a general overview of the Altera IP core design flow to help you quickly get started with any Altera IP core. The Altera IP library is installed as part of the Quartus II installation process. You can select and parameterize any Altera IP core from the library. Altera provides an integrated parameter editor that allows you to customize IP cores to support a wide variety of applications. The parameter editor guides you through the setting of parameter values and selection of optional ports.

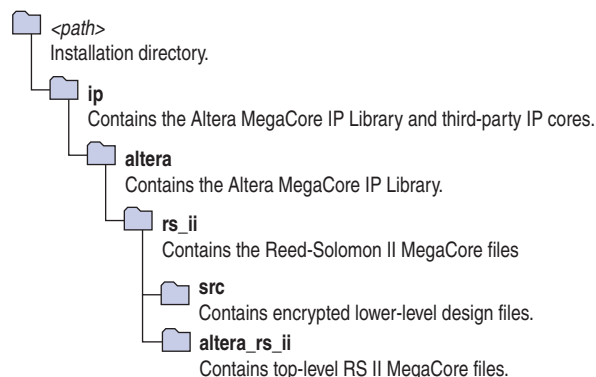
The following sections describe the general installation, design flow, evaluation, and production use of Altera IP cores.

### Installation and Licensing

The Altera IP Library is distributed with the Quartus II software and downloadable from the Altera website, [www.altera.com](http://www.altera.com).

Figure 2–1 shows the directory structure after you install an Altera IP core, where *<path>* is the installation directory. The default installation directory on Windows is `C:\altera\<version number>`; on Linux it is `/opt/altera<version number>`.

**Figure 2–1. Directory Structure**



You can evaluate an IP core in simulation and in hardware before you purchase a license. For most Altera IP cores, you can use Altera’s free OpenCore Plus evaluation feature for this purpose. Some Altera IP cores do not require the use of this special feature for evaluation. You can evaluate the IP core until you are satisfied with its functionality and performance. You must purchase a license for the IP core when you want to take your design to production.

After you purchase a license for an Altera IP core, you can request a license file from the Altera website at [www.altera.com/licensing](http://www.altera.com/licensing) and install it on your computer. When you request a license file, Altera emails you a **license.dat** file. If you do not have internet access, contact your local Altera representative.



For additional information about installation and licensing, refer to *Altera Software Installation and Licensing*.



## Evaluating an IP Core

The Altera IP library contains both free and individually licenced IP cores. With the Altera free OpenCore Plus evaluation feature, you can evaluate separately licenced IP cores in the following ways prior to purchasing a production license:

- Simulate the behavior of an Altera IP core in your system using the Quartus II software and Altera-supported VHDL and Verilog HDL simulators.
- Verify the functionality of your design and evaluate its size and speed quickly and easily.
- Generate device programming files for designs that include IP cores. These files are time-limited under the OpenCore Plus evaluation program.
- Program a device and verify your design in hardware.

### Open Core Plus Time-Out Behavior

OpenCore Plus hardware evaluation supports the following two operation modes:

- *Untethered*—the design runs for a limited time.
- *Tethered*—requires a connection between your board and the host computer. If all Altera IP cores in a design support tethered mode, the device can operate for a longer time or indefinitely.

All IP cores in a device time out simultaneously when the most restrictive evaluation time is reached. If there is more than one IP core in a design, a specific IP core's time-out behavior may be masked by the time-out behavior of the other IP cores.



For IP cores, the untethered time-out is 1 hour; the tethered time-out value is indefinite.

Your design stops working after the hardware evaluation time expires.



The Quartus II software uses OpenCore Plus Files (.ocp) in your project directory to identify your use of the OpenCore Plus evaluation program. After you activate the feature, do not delete these files.



For information about the OpenCore Plus evaluation program, refer to [AN320: OpenCore Plus Evaluation of Megafunctions](#).

## MegaWizard Plug-In Manager Design Flow

The MegaWizard™ Plug-in Manager flow allows you to customize a RS II MegaCore function, and manually integrate the MegaCore function variation in a Quartus II design.

### Specifying Parameters

To specify parameters with the MegaWizard Plug-In Manager, follow these steps:

1. Create a Quartus II project using the **New Project Wizard** available from the File menu.
2. In the Quartus II software, launch the **MegaWizard Plug-in Manager** from the Tools menu, and follow the prompts in the MegaWizard Plug-In Manager interface to create or edit a custom IP core variation.
3. To select a specific Altera IP core, click the IP core in the **Installed Plug-Ins** list in the MegaWizard Plug-In Manager.

For example, to specify a Reed-Solomon II MegaCore function, click **Installed Plug-Ins > DSP > Error Detection/Correction > Reed Solomon II <version>**.

4. Verify that the device family is the same as you specified in the **New Project Wizard**.
5. Select the top-level output file type for your design; the MegaWizard Plug-In Manager supports VHDL and Verilog HDL.
6. Specify the top-level output file name for your MegaCore function variation and click **Next** to launch the IP Toolbench.
7. Specify the parameters on the **Parameter Settings** pages. For detailed explanations of these parameters, refer to [“Parameter Settings” on page 2–5](#).
8. Click the **Finish** button. The generation phase may take several minutes to complete. The generation progress and status is displayed in a report window. The parameter editor generates the top-level HDL code for your IP core, a Quartus II IP file (**.qip**) file containing all of the necessary assignments and information required to process the IP core in the Quartus II Compiler, and a simulation directory which includes files for simulation.

You can now integrate your custom IP core instance in your design, simulate, and compile. While integrating your IP core instance into your design, you must make appropriate pin assignments. You can create virtual pins for top-level signals if you want to avoid making specific pin assignments while simulating and not ready to map the design to hardware.



For information about the Quartus II software, including virtual pins and the MegaWizard Plug-In Manager, refer to Quartus II Help.

## Simulating the Design

You can simulate your IP core variation with the functional simulation model. The functional simulation model and testbench files are generated in your project directory or a designated directory.



For more information about simulating Altera IP cores, refer to *Simulating Altera IP in Third-Party Simulation Tools* and *Simulating Designs with EDA Tools* in volume 3 of the *Quartus II Handbook*.

## Compiling the Design and Programming a Device

After using the MegaWizard Plug-In Manager to define and instantiate your IP core, you must compile your design to create programming files to configure the FPGA.

Some Altera IP cores require that you apply constraints before compilation. These constraint files make pin assignments and ensure that your IP core instance meets design timing requirements.

After applying the constraint files if appropriate for your IP core, click **Start Compilation** on the Processing menu in the Quartus II software to compile your design. After successfully compiling your design, program the targeted Altera device with the Programmer and verify the design in hardware.

## Parameter Settings

This section describes the parameters available in the RS II MegaCore function.

For information about using the parameter editor, refer to “[MegaWizard Plug-In Manager Design Flow](#)” on page 2-3.

Table 2-1 lists the parameter settings for the RS II MegaCore function.

**Table 2-1. Parameter Settings for RS II MegaCore Function**

Parameter	Legal Values	Default Value	Description
Reed-Solomon	Encoder or Decoder	Encoder	Specifies an encoder or a decoder.
Number of symbols per codeword	204–255	255	Specifies the total number of symbols per codeword ( $N$ ).
Number of check symbols per codeword	2–66	16	Specifies the number of check symbols per codeword ( $R$ ).
Field Polynomial	Any valid polynomial (1)	285	Specifies the primitive polynomial defining the Galois field.
Number of channels	1, 2, 8, 16	1	Specifies the number of input channels to process. The channel pattern is fixed.

**Note to Table 2-1:**

(1) The parameter editor allows you to select only legal values. If you cannot find your intended field polynomial, contact Altera MySupport.

The RS II MegaCore function has the following fixed value parameters:

- Number of bits per symbol = 8
- Number of symbols per beat = 1
- First root of generator polynomial = 0
- Root spacing in generator polynomial = 1

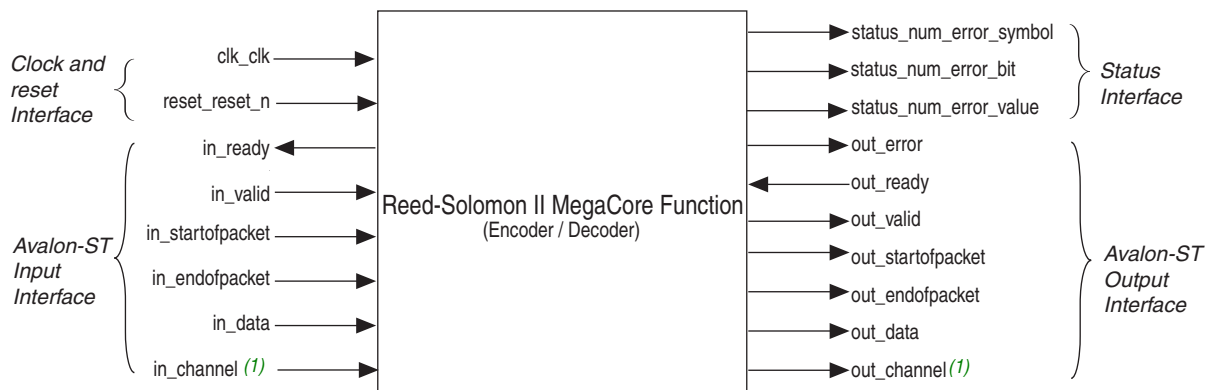
This chapter describes in detail about the RS II MegaCore function, its architecture, interfaces, and interface signals.

### Architecture

The RS II MegaCore function can act as an encoder or a decoder. The encoder receives data packets and generates the check symbols, while the decoder detects and corrects errors.

Figure 3–1 shows a high-level block diagram of the RS II MegaCore function.

**Figure 3–1. RS II Block Diagram**



**Note to Figure 3–1:**

(1) The `in_channel` and `out_channel` ports are available only when you configure the IP core to support multi-channels.

### Interfaces

The RS II MegaCore function includes the following interfaces:

- Avalon® Streaming (Avalon-ST) input and output interfaces
- Clock and reset interfaces
- Status interface

## Avalon-ST Input and Output Interfaces

The input and output interfaces of the MegaCore function implement the Avalon-ST protocol, which is a unidirectional flow of data. The input interface is an Avalon-ST sink and the output interface is an Avalon-ST source. The number of bits per symbol on these interfaces is fixed to 8; the number of symbols per beat is 1. The ready latency on the RS II Avalon-ST input interface is 0. The RS II Avalon-ST interface supports packet transfers with packets interleaved across multiple channels. The Avalon-ST interface inherently synchronizes multi-channel designs, which allows you to achieve efficient, time-multiplexed implementations without having to implement complex control logic.

The RS II Avalon-ST interface also supports backpressure, which is a flow control mechanism, where a sink can signal to a source to stop sending data.



For more information about the Avalon-ST interface, refer to the [Avalon Interface Specifications](#).

## Clock and Reset Interfaces

The clock and reset interfaces drive or receive the clock and reset signal to synchronize the Avalon-ST interfaces and provide reset connectivity. You must deassert the reset signal synchronously to the clock signal.

## Status Interface

The status interface is a conduit interface that consists of three error status signals for a codeword. The decoder obtains the error value, total number of error symbols, and total number of error bits in a codeword from the status signals.

## RS II Encoder

When the encoder receives data symbols, it generates check symbols for a given codeword and sends the input codeword together with the check symbols to the output interface. The encoder backpressures the upstream component when it generates the check symbols.

Figure 3–2 shows how a codeword is encoded.

**Figure 3–2. Reed-Solomon II Encoding**

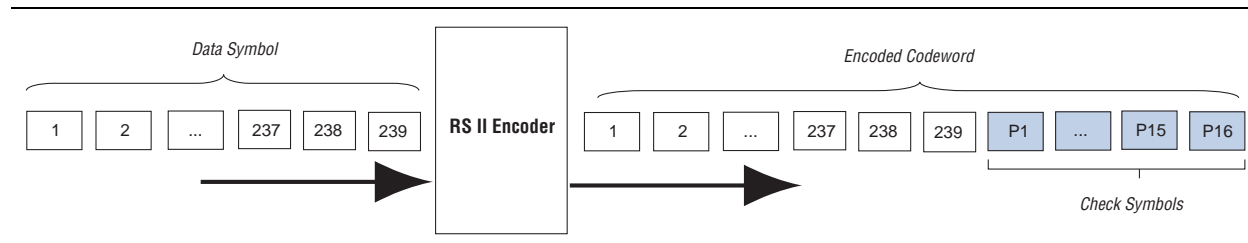
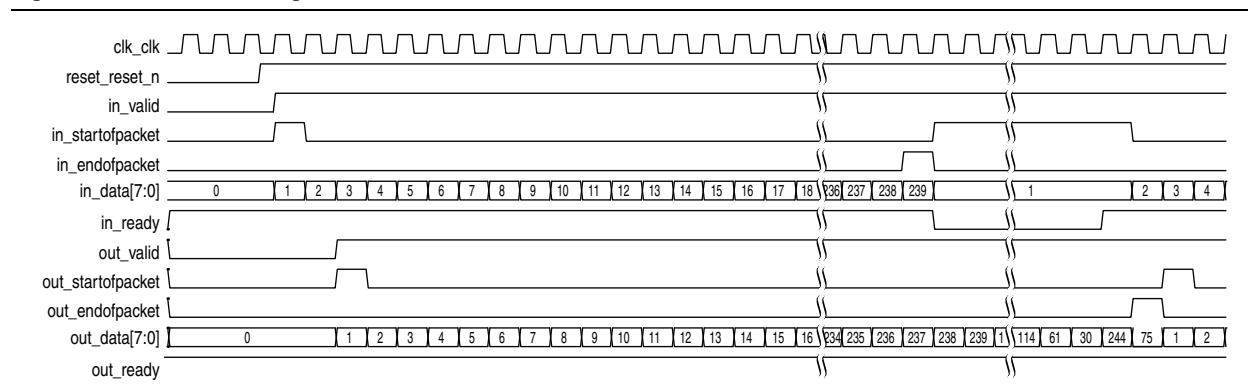


Figure 3–3 shows the timing diagram of the RS II encoder with one channel.

**Figure 3–3. Encoder Timing—One Channel**



The `in_startofpacket` signal starts a codeword; the `in_endofpacket` signals its termination. An asserted `in_valid` signal indicates valid data. The `in_startofpacket` signal is only valid when you assert the `in_valid` signal. For a 1-channel codeword, assert the `in_startofpacket` and `in_endofpacket` signals for one clock cycle.

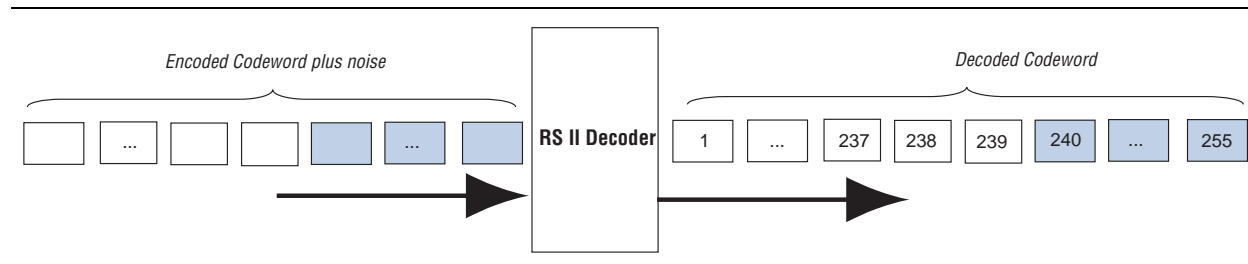
The encoder starts backpressure by deasserting the `in_ready` signal when it receives the `in_endofpacket` signal. During this time, the encoder signals that it cannot accept more incoming symbols and generates the check symbols for the current codeword. The IP core does not verify if the number of symbols ( $N$ ) exceeds the maximum symbols per codeword. You must ensure that the codeword sent to the core has a valid  $N$ . The `reset_reset_n` signal is active low and you can assert this signal asynchronously. However, you have to deassert the `reset_reset_n` signal synchronously with the `clk_clk` signal.

## RS II Decoder

When the decoder receives the encoded codeword, it uses the check symbols to detect errors, and corrects them.

Figure 3–4 shows how a codeword is decoded.

**Figure 3–4. RS II Decoding**



The received encoded codeword may differ from the original codeword due to the noise in the channel. The decoder detects errors using several polynomials to locate the error location and the error value.

For more information about using polynomials to locate errors, refer to “RS Decoding” on page A-3.

When the decoder obtains the error location and value, the decoder corrects the errors in a codeword, and sends the codeword to the output. As the number of errors increases, the decoder gets to a stage where it can no longer correct but only detect errors, at which point the decoder asserts the `out_error` signal.

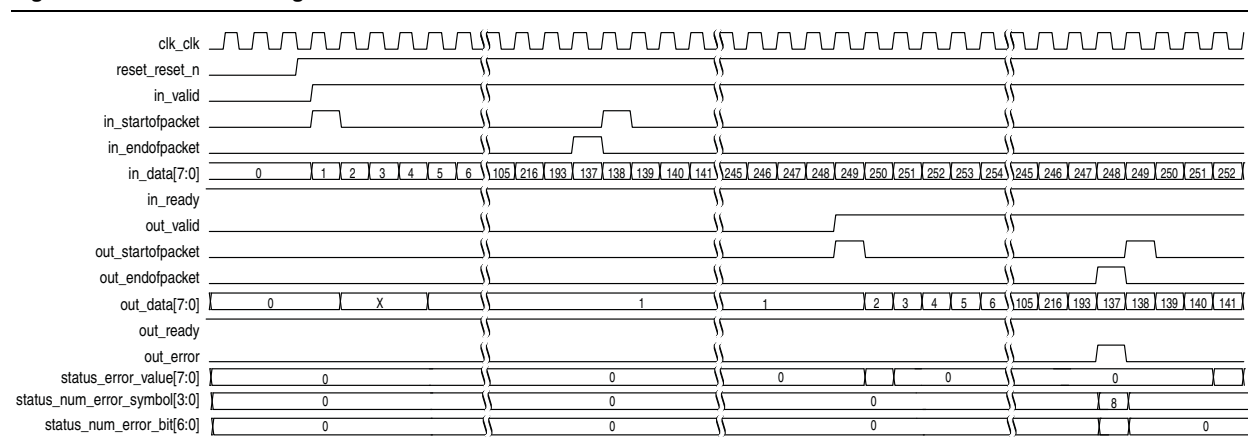
Table 3-1 lists how the decoder corrects and detects errors depending on the number of check symbols ( $R$ ).

**Table 3-1. Decoder Detection and Correction**

Number of Errors	Description
$\text{Errors} \leq R/2$	Decoder detects and corrects errors.
$R/2 \leq \text{errors} \leq R$	Decoder asserts error signal and can only detect errors.
$\text{Errors} > R$	Unpredictable results.

Figure 3-5 shows the timing diagram of the RS II decoder with one channel.

**Figure 3-5. Decoder Timing—One Channel**



The codeword starts when you assert the `in_valid` signal and the `in_startofpacket` signal. The decoder accepts the data at `in_data` as valid data. The codeword ends when you assert the `in_endofpacket` signal. For a 1-channel codeword, assert the `in_startofpacket` and `in_endofpacket` signals for one clock cycle.

When the decoder deasserts the `in_ready` signal, the decoder cannot process any more data until the decoder asserts the `in_ready` signal again.

At the output, the operation is identical. When the decoder asserts the `out_valid` signal and the `out_startofpacket` signal, the decoder provides valid data on `out_data`. The decoder asserts the `out_startofpacket` signal and the `out_endofpacket` signal to indicate the start and end of a codeword. The decoder automatically detects and corrects errors in a codeword and asserts the `out_error` signal when it encounters a non-correctable codeword.



## Multi-Channel Codeword

The RS II MegaCore function processes multiple input channels simultaneously. The IP core receives codeword in a fixed pattern. Symbols coming in through the channels are interlaced. The RS II MegaCore function samples the first symbol of channel one on the first rising clock edge, then the first symbol of channel two on the second rising clock edge, and so on. Both information and check symbols are output in the same sequence.

Figure 3–6 shows a codeword with  $k$  channels and  $N$  symbols. The channel signal indicates the channel associated to the current symbol. The channel sequence is fixed. `startofpacket` indicates the first symbol of a codeword per channel. For a  $k$ -channel codeword, `startofpacket` must be high for  $k$  consecutive cycles. `endofpacket` indicates the last symbol of a codeword per channel. For a  $k$ -channel codeword, `endofpacket` must be high for  $k$  consecutive cycles.



`startofpacket` and `endofpacket` governs the number of symbols per codeword,  $N$ . The core does not verify if  $N$  exceeds the maximum symbols per codeword. The core also does not verify the channel or data pattern. You must ensure that the codeword sent to the core has a valid  $N$  and a valid pattern.

Figure 3–6. Codeword for  $k$  Channels and  $N$  Symbols

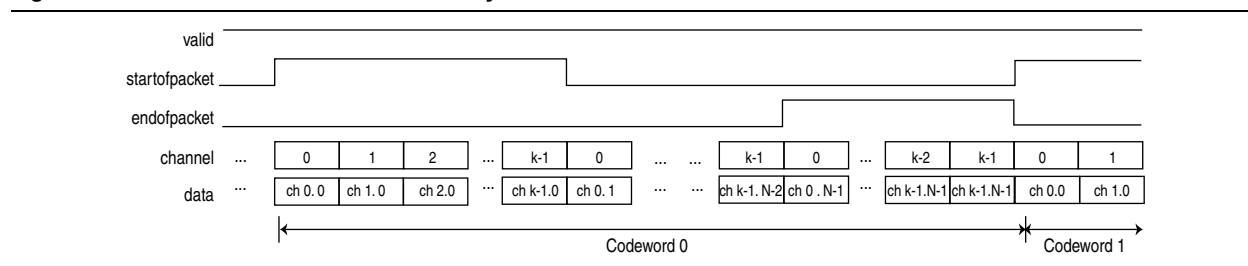


Figure 3–7 shows the timing diagram of an encoder with two channels. For a 2-channel codeword, the encoder asserts the `in_startofpacket` and `in_endofpacket` signals for two consecutive cycles.

Figure 3–7. Encoder Timing—Two Channels

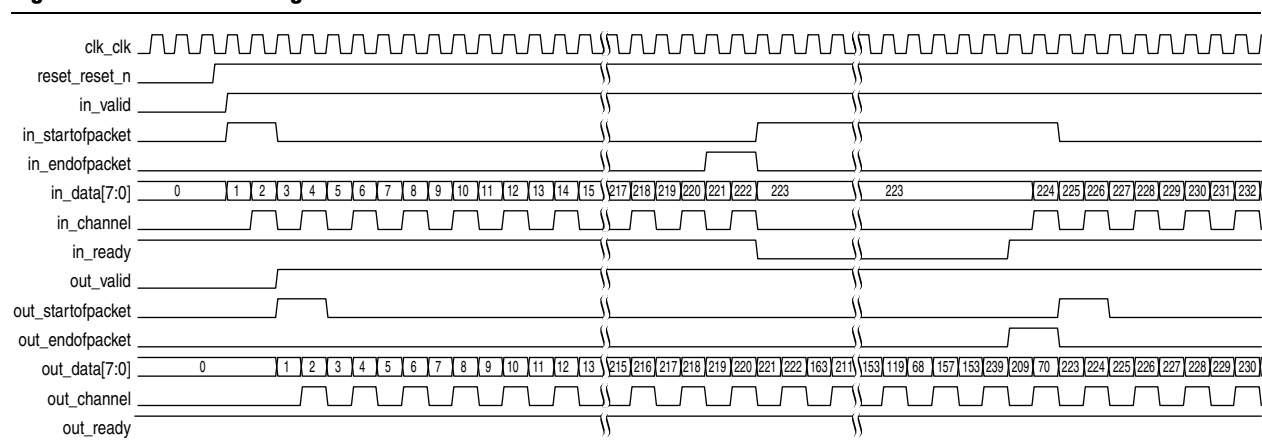
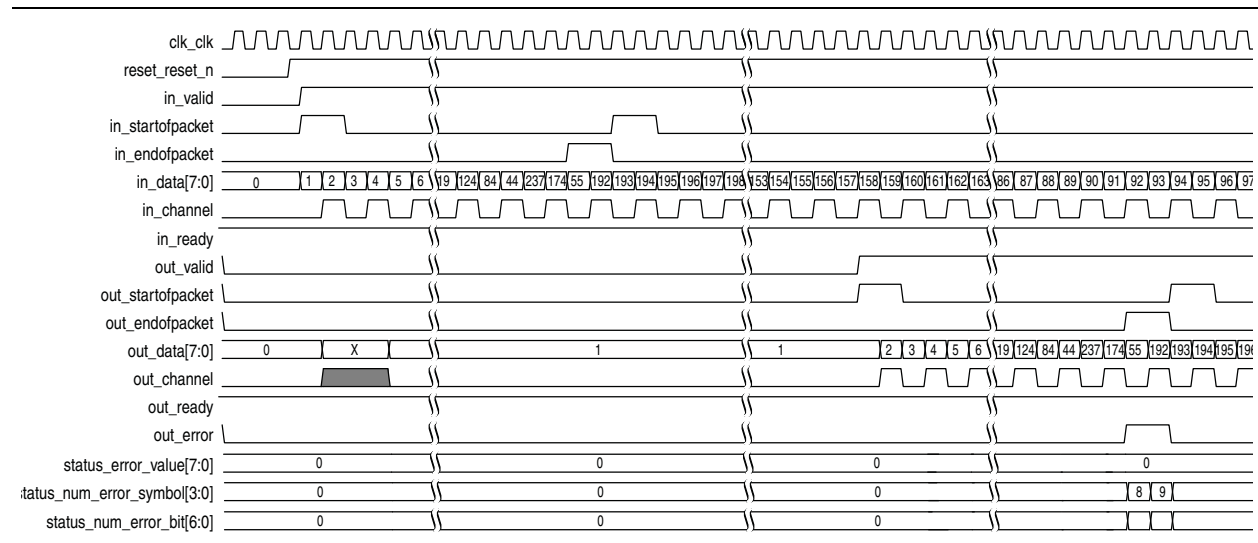


Figure 3–8 shows the timing diagram of the RS II decoder with two channels. For a 2-channel codeword, the decoder asserts the `in_startofpacket` and `in_endofpacket` signals for two consecutive cycles.

**Figure 3–8. Decoder Timing—Two Channels**



## Signals

Table 3–2 lists the clock and reset signals.

**Table 3–2. Clock and Reset Signals**

Name	Avalon-ST Type	Direction	Description
clk_clk	clk	Input	clk_clk is the main system clock. The whole MegaCore function operates on the rising edge of clk_clk.
reset_reset_n	reset_n	Input	An active low signal that resets the entire system when asserted. You can assert this signal asynchronously. However, you must deassert it synchronous to the clk_clk signal. When the MegaCore function recovers from reset, ensure that the data received by the MegaCore function is a complete packet. Altera recommends that you stop the datapath by not sending anymore valid data before you reset the MegaCore function and send the next complete packet.

Table 3–3 lists the signals on the RS II Avalon-ST input and output interfaces.

**Table 3–3. RS II Avalon-ST Input and Output Interface Signals (Part 1 of 2)**

Name	Avalon-ST Type	Direction	Description
in_ready	ready	Output	Data transfer ready signal to indicate that the sink is ready to accept data. The sink interface drives the in_ready signal to control the flow of data across the interface. The sink interface captures the data interface signals on the current clk rising edge.
in_valid	valid	Input	Data valid signal to indicate the validity of the data signals. When you assert the in_valid signal, the Avalon-ST data interface signals are valid. When you deassert the in_valid signal, the Avalon-ST data interface signals are invalid and must be disregarded. You can assert the in_valid signal whenever data is available, however the sink only captures the data from the source only when the MegaCore function asserts the in_ready signal.
in_data[]	data	Input	Data input for each codeword, symbol by symbol. Valid only when you assert the in_valid signal.
in_channel	channel	Input	Specifies the channel number for data being transferred on the current cycle. The in_channel signal is available only when you configure the MegaCore function to support multi-channels.
in_startofpacket	sop	Input	Start of packet (codeword) signal.
in_endofpacket	eop	Input	End of packet (codeword) signal.
out_startofpacket	sop	Output	Start of packet (codeword) signal. This signal indicates the codeword boundaries on the in_data[] bus. When the MegaCore function drives this signal high, it indicates that the start of packet is present on the in_data[] bus. The MegaCore function asserts this signal on the first transfer of every codeword.
out_endofpacket	eop	Output	End of packet (codeword) signal. This signal indicates the packet boundaries on the in_data[] bus. When the MegaCore function drives this signal high, it indicates that the end of packet is present on the in_data[] bus. The MegaCore function asserts this signal on the last transfer of every packet.

**Table 3–3. RS II Avalon-ST Input and Output Interface Signals (Part 2 of 2)**

Name	Avalon-ST Type	Direction	Description
out_ready	ready	Input	Data transfer ready signal to indicate that the downstream module is ready to accept data. The source provides new data (if available) when you assert the <code>out_ready</code> signal and stops providing new data when you deassert the <code>out_ready</code> signal. If the source is unable to provide new data, it deasserts <code>out_valid</code> for one or more clock cycles until it is prepared to drive valid data interface signals.
out_valid	valid	Output	Data valid signal. The MegaCore function asserts the <code>out_valid</code> signal high, whenever there is a valid output on <code>out_data</code> ; the MegaCore function deasserts the signal when there is no valid output on <code>out_data</code> .
out_data	data	Output	The <code>out_data</code> signal contains decoded output when the MegaCore function asserts the <code>out_valid</code> signal. The corrected symbols are in the same order that they were entered.
out_channel	channel	Output	Specifies the channel whose result is presented at <code>out_data</code> . The <code>out_channel</code> signal is available only when you configure the MegaCore function to support multi-channels.
out_error	error	Output	Indicates non-correctable codeword (decoder only). This signal is valid when the MegaCore function asserts <code>out_endofpacket</code> .

Table 3–4 lists the status interface signals.



The value for these status interface signals are valid when the codeword contains errors that can be corrected by the decoder. Otherwise, these signals contain any value.

**Table 3–4. Status Interface Signals**

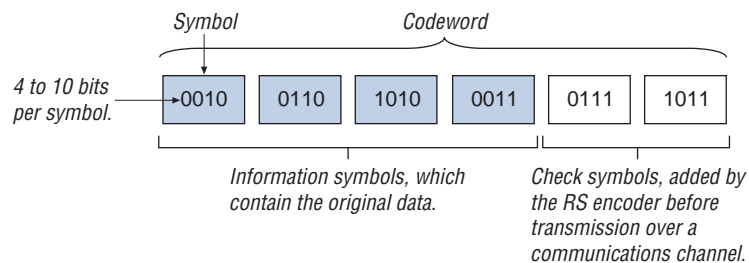
Name	Avalon-ST Type	Direction	Description
status_num_error_symbol	conduit	Output	Number of error symbols in a codeword. This signal is valid when the MegaCore function asserts the <code>out_endofpacket</code> .
status_num_error_bit	conduit	Output	Number of error bits in a codeword. This signal is valid when the MegaCore function asserts the <code>out_endofpacket</code> .
status_error_value	conduit	Output	Error correction value for every valid data symbol.

## RS Encoding

To use RS codes, a data stream is first broken into a series of codewords. Each codeword consists of several information symbols followed by several check symbols (also known as parity symbols or redundant symbols). Symbols can contain an arbitrary number of bits. In an error correction system, the encoder adds check symbols to the data stream prior to its transmission over a communication channel. When the decoder receives the data, the decoder checks for and corrects any errors.

Figure A-1 shows an example of a RS codeword.

**Figure A-1. RS Codeword Example**



RS codes are described as  $(N,K)$ , where  $N$  is the total number of symbols per codeword and  $K$  is the number of information symbols.  $R$  is the number of check symbols ( $N - K$ ). Errors are defined on a symbol basis. Any number of bit errors within a symbol is considered as only one error.

A Reed-Solomon code is characterized by the following two polynomials:

- Field polynomial
- Generator polynomial

## Field polynomial

The field polynomial is based on finite-field (Galois field) arithmetic, of which any arithmetic operation (addition, subtraction, multiplication, and division) on a field element gives a result that is an element of the field. The size of the Galois field is determined by the number of bits per symbol—specifically, the field has  $2^m$  elements, where  $m$  is the number of bits per symbol. A specific Galois field is defined by a polynomial, which is user-defined for the RS II MegaCore function.

## Generator polynomial

The generator polynomial defines how the check symbols are generated. The maximum number of symbols in a codeword is limited by the size of the finite field to  $2^m - 1$ .

The following equation represents the generator polynomial of the code:

$$g(x) = \prod_{i=0}^{R-1} (x - \alpha^{a \cdot i + i_0})$$

where:

$i_0$  is the first root of the generator polynomial

$a$  is the rootspace

$R$  is the number of check symbols

$\alpha$  is a root of the polynomial.

## Shortened Codewords

The RS II MegaCore function supports shortened codewords. A shortened codeword contains fewer symbols than the maximum value of  $N$ , which is  $2^m - 1$ . A shortened codeword is mathematically equivalent to a maximum-length code with the extra data symbols at the start of the codeword set to 0.

For example, (204,188) is a shortened codeword of (255,239). Both of these codewords use the same number of check symbols, 16.

To use shortened codewords with the Altera RS II encoder and decoder, use the parameter editor to set the codeword length to the correct value, in the example, 204.

## RS Decoding

The input codeword represents the received codeword,  $R(x)$ , which consists of the transmitted codeword  $T(x)$  and the error introduced during transmission,  $E(x)$ . The received codeword is represented in the following equation:

$$R(x) = T(x) + E(x)$$

$R(x)$  can also be represented in a polynomial form:

$$R(x) = A_{N-1}x^{N-1} + A_{N-2}x^{N-2} + \dots + A_1x + A_0$$

where,  $A$  is the input symbol ( $A_{N-1}$  is the first symbol) and  $N$  is the codeword length

The decoder performs the following steps to decode a received codeword:

1. Generates the syndrome polynomial.
2. Generates two error polynomials based on the syndrome polynomial.
3. Solves the two error polynomials to locate errors and calculate error values.

### Syndrome Polynomial

The syndrome generator generates the syndrome polynomial in the first step of the decoding process.

The equation of the syndrome polynomial is given by,

$$S(x) = S_{2t-1}x^{2t-1} + \dots + S_1x + S_0$$

where  $2t$  = Number of check symbols

The syndrome generator uses the Horner's method to generate the syndrome polynomial.

### Error Polynomials

After generating the syndrome polynomial, the next step in the decoding process is to use the Berlekamp-Massey (BM) algorithm to find the following two error polynomials:

- Error locator polynomial,  $\Lambda(x)$
- Error evaluator polynomial,  $\Omega(x)$

The equation of the error locator polynomial is given by,

$$\Lambda(x) = \Lambda_t x^t + \Lambda_{t-1} x^{t-1} + \dots + \Lambda_1 x + 1$$

The equation of the error evaluator polynomial is given by,

$$\Omega(x) = \Omega_{t-1} x^{t-1} + \dots + \Omega_1 x + \Omega_0$$

where  $t$  = Number of check symbols/2

The BM algorithm is a technique of forming an initial error locator polynomial, followed by multiple iterations of the same polynomial to improve and eventually identify the correct polynomial.

## Error Location and Error Value

After the decoder forms the error locator polynomial, the decoder solves the polynomial to find the error location. The decoder determines the roots of the polynomial through a trial and error method, known as the Chien search. The decoder substitutes every possible root  $\alpha^{-e}$ , where  $e$  represents the location in a codeword, into the error locator polynomial. A zero result indicates that the corresponding location contains an error.

After the decoder obtains the error locations, the decoder calculates the error values using the Forney's equation,

$$Y(j) = X_j \frac{\Omega(X^{-1}j)}{\Lambda'(X^{-1}j)}$$

where  $\Lambda'(X^{-1}j)$  is the derivative of  $\Lambda(x)$  for  $x = X^{-1}j$



This chapter provides additional information about the document and Altera.

## Document Revision History

The following table shows the revision history for this document.

Date	Version	Changes
May 2011	2.0	<ul style="list-style-type: none"> <li>Updated <a href="#">Chapter 1, About This MegaCore</a> with new device family support.</li> <li>Updated <a href="#">Chapter 3, Functional Description</a> with new status ports and timing diagrams.</li> </ul>
December 2010	1.0	Initial release.

## How to Contact Altera

To locate the most up-to-date information about Altera® products, refer to the following table.

Contact (1)	Contact Method	Address
Technical support	Website	<a href="http://www.altera.com/support">www.altera.com/support</a>
Technical training	Website	<a href="http://www.altera.com/training">www.altera.com/training</a>
	Email	<a href="mailto:custrain@altera.com">custrain@altera.com</a>
Product literature	Website	<a href="http://www.altera.com/literature">www.altera.com/literature</a>
Non-technical support (General) (Software Licensing)	Email	<a href="mailto:nacomp@altera.com">nacomp@altera.com</a>
	Email	<a href="mailto:authorization@altera.com">authorization@altera.com</a>








### Note to Table:

(1) You can also contact your local Altera sales office or sales representative.

## Typographic Conventions

The following table shows the typographic conventions this document uses.

Visual Cue	Meaning
<b>Bold Type with Initial Capital Letters</b>	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, <b>Save As</b> dialog box. For GUI elements, capitalization matches the GUI.
<b>bold type</b>	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, <b>\qdesigns</b> directory, <b>D:</b> drive, and <b>chiptrip.gdf</b> file.
<i>Italic Type with Initial Capital Letters</i>	Indicate document titles. For example, <i>Stratix IV Design Guidelines</i> .
<i>italic type</i>	Indicates variables. For example, $n + 1$ . Variable names are enclosed in angle brackets (< >). For example, <file name> and <project name>.pdf file.

Visual Cue	Meaning
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
“Subheading Title”	Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, “Typographic Conventions.”
Courier type	Indicates signal, port, register, bit, block, and primitive names. For example, <code>data1</code> , <code>tdi</code> , and <code>input</code> . The suffix <code>n</code> denotes an active-low signal. For example, <code>resetn</code> . Indicates command line commands and anything that must be typed exactly as it appears. For example, <code>c:\qdesigns\tutorial\chiptrip.gdf</code> . Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword <code>SUBDESIGN</code> ), and logic function names (for example, <code>TRI</code> ).
	An angled arrow instructs you to press the Enter key.
1., 2., 3., and a., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■ ■	Bullets indicate a list of items when the sequence of the items is not important.
	The hand points to information that requires special attention.
	A question mark directs you to a software help system with related information.
	The feet direct you to another document or website with related information.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
	A warning calls attention to a condition or possible situation that can cause you injury.
	The envelope links to the <a href="#">Email Subscription Management Center</a> page of the Altera website, where you can sign up to receive update notifications for Altera documents.